However, readability is more than just programming style..  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation.  
 Auxiliary tasks accompanying and related to programming include analyzing requirements, testing, debugging (investigating and fixing problems), implementation of build systems, and management of derived artifacts, such as programs' machine code.  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
 Allen Downey, in his book How To Think Like A Computer Scientist, writes:  
 Many computer languages provide a mechanism to call functions provided by shared libraries.  
Integrated development environments (IDEs) aim to integrate all such help.  
Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability.  
 After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug.  
 Programs were mostly entered using punched cards or paper tape.  
However, because an assembly language is little more than a different notation for a machine language, two machines with different instruction sets also have different assembly languages.  
 Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications.  
 Code-breaking algorithms have also existed for centuries.